PARALLEL LIGHT SPEED LABELING: THE WORLD'S FASTEST CONNECTED COMPONENT LABELING ALGORITHM FOR MULTICORE PROCESSORS

Laurent Cabaret, Lionel Lacassagne, Daniel Etiemble

Laboratoire de Recherche en Informatique
INRIA - Univ. Paris Sud F-91405 Orsay, France
firstname.name@lri.fr

ABSTRACT

The paper introduces the parallel version of the Light Speed Labeling (LSL) and compares it with the parallel versions of the competitors. A benchmark shows that the parallel Light Speed Labeling is $\times 1.9$ faster than all the other algorithms for random images on average. This factor reach $\times 3.6$ for structured random images. More important, we show that thanks to its run-based processing (segments), LSL is intrinsically more efficient than all pixel-based algorithms.

Index Terms— Connected component labeling, parallel algorithms, bounding box, Union-Find, transitive closure.

Introduction

Binary Connected Component Labeling (CCL) algorithms deal with graph coloring and transitive closure computation. CCL algorithms play a central part in machine vision, because they often constitute a mandatory step between low-level image processing (filtering) and high-level image processing (recognition, decision). Designing a parallel algorithm is challenging both from considering the overwhelming literature of sequential algorithms and the fact that the parallelization of the fastest sequential algorithm may not lead to the fastest parallel one. Moreover if the speedup is an interesting metric to evaluate the parallelization and the scalability, this is the execution time (here in cycles per pixel) that matters.

In prior work, we have presented the sequential version of the Light Speed Labeling [1, 2] and compared it to State-of-the-Art competitors [3]. Our contribution was to provide an exhaustive benchmark procedure and to focus on intrinsic performance of LSL. In [4] the authors present a new implementation of specific embedded multicore system that does not scale ($\times 11.38$ on 64 cores). The contribution of this paper is to introduce the parallel version of LSL for multicore general purpose processor with OpenMP parallelization. A multi-step benchmark focus on the fact that LSL is intrinsically more efficient and faster than all other pixel-based algorithms, thanks to it run-based approach and RLE compression.

This paper is organized as follows: the first section introduces the parallel versions of CCL algorithms: their structure modification from the sequential version. The second section is the extension of the benchmark procedure proposed in [3] to parallel versions: random images with varying density *and* granularity.

1. ALGORITHMS

Usually, CCL algorithms have two passes: the first pass provides a temporary labeling and the associated equivalences building. The inter-pass algorithm is a graph transitive closure (TC) that provides a unique label to each connected component and the second pass is the final labeling that applies the equivalence table to the temporary labels.

An other important point to consider when designing a CCL algorithm is its usage. It bridges semantic visual levels in providing bounding rectangle and the first order statistical moments. So if a standalone CCL algorithm can be considered at first step, the couple "CCL+features computation" is the procedure to be actually evaluated at end. The typical step is to do the first labeling on the whole image, then solve the equivalences (EQ), do the final labeling and finally perform the features computation (FC) step (fig. 1, left). But if the FC is performed *on-the-fly* during the first labeling, the final labeling is no more required (fig. 1, center). In that case the first labeling step consists in a line-labeling function and a line FC function that are applied to the whole image. The procedure to solve the equivalences is then modified to also update the features (algo. 1). Thus, the two passes can be reduced to only one pass where FC is done *on-the-fly*. With that modification, all two-pass algorithms become one-pass algorithms.

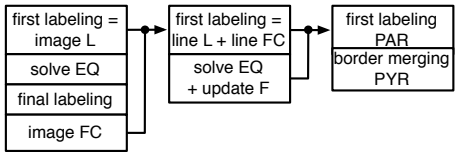

Fig. 1. sequential (left and center) and parallel (right) CCL synopsis

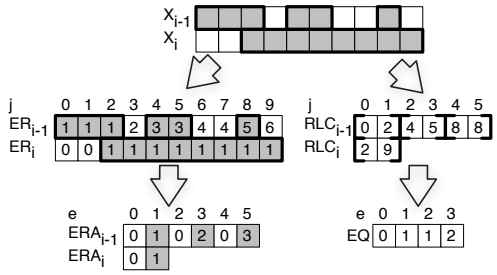

Fig. 2. LSL tables

The specificity of LSL is to be run-based and to use a line-relative labeling (fig. 2). From two consecutive lines X_i, X_{i-1}, two relative labelings are produced where runs (or segments) have odd numbers. In the same way, the associated *run-length-codings* are produced (tables RLC_i, RLC_{i-1}). The table ERA_{i-1} holds the translation between *Relative* and *Absolute* labels. To find out the labels of the previous line that are connected to the current segment, one has to read in table ER_{i-1} at the position given by RLC_i the value of relative labels and translates them into absolute labels to update the equivalence table EQ. Details about sequential implementation of the LSL are available in [2].

Algorithm 1: solve equivalences & update features

```
for each e ∈ [1 : ne] do
    r ← T[T[e]] // root of the tree
    T[e] ← r
    F[r] ← F[r] ∪ F[e] // features update
```

In order to figure out what part of the algorithm is important (that makes one algorithm to be faster than other ones), we modified the LSL: there are still the STD (standard) and RLE (RLE compression) versions, but the equivalence management has been replaced by either a classic Union-Find (as in Rosenfeld's one) or Suzuki's method [5]. The competitor algorithms we have parallelized are:

- $Rosenfeld$: original Rosenfeld [6] algorithm improved with Decision Tree (DT) and Path Compression (PC)
- $Suzuki$: $Rosenfeld$ mask with Suzuki tables management [5] improved with DT,
- RCM: pixel-based algorithm with Suzuki management and DT [7],
- HCS_2: block-based algorithm with Suzuki management [8],
- $Grana$: block-based algorithm with 128-stage DT, using Suzuki management [9],
- HCS: run-based algorithm with Suzuki management [10],
- LSL: run-based algorithm with either Rosenfeld or Suzuki management, with two variants: LSL_{STD} (standard version) and LSL_{RLE} (version with compression).

1.1. Parallelization

First parallel labeling: For cache-awareness, the image is split into p horizontal strips, with p the number of cores. The first step of the parallel labeling (fig. 1, right) correspond to the combination of the two steps of the sequential labeling. This step is done in parallel on each strip of the image thanks to OpenMP. Instead of starting label number at 0 in each strip that will required to renumber of all the labels by adding an offset (like in [4]) and a synchronization, the numbering starts to the sum of the max number of temporary labels possible for each strip of size $h \times w$ that is equal to $\frac{h+1}{2} \times \frac{w+1}{2}$ for 8-connectivity.

Border merging : There are three strategies for merging the borders. The first one is sequential and inefficient: each border is sequentially processed with the same function that is used within the first labeling. The second one is parallel and false – from a concurrency point of view – as a given label (and its associated features) must be modified by exactly one actor. The wrong modification here would be to use mutexes (or semaphores or locks) to serialize updates which would result in many synchronizations and data races leading to poor parallel performance and "global" serialization. The third one is pyramidal: for each level of the tree (Fig. 3), the merges can be done in parallel. This is allowed as such a division ensures that, at each step, the strip containing a given label, will be merged with only one strip at a time. Concurrency issues should be addressed by algorithm modification not by programming.

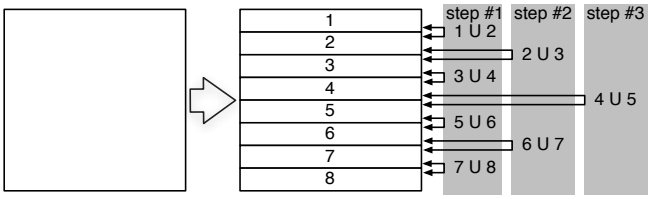

Fig. 3. Pyramidal merging of borders

2. BENCHMARK

Procedure: For the benchmark, we use a 2×12-core Ivy-Bridge Xeon E5-2695 v2 at 2.4 GHz running Red-Hat Linux with ICC 14. The same benchmark protocol as described in [3] is used: random images with density d varying from 0% to 100% with a step of 0.2%, and granularity g varying from 1 to 16 with a step of 1. In order to have realistic speedups and results we limit the image size to 2048×2048, first because it is close to the biggest image size that a cameras can acquire (between full-HD 1920×1080 and 4K), and because, on such a machine, the amount of pixels *per* core is close to the amount of pixels of a 512×512 image classically used for the sequential algorithm. With higher image size, the merge

time will be very small and thus the speedup would be too optimistic. The following results are for three granularities. First g=1 as it is the worst configuration for LSL, second g=4 as it was the turning point for sequential algorithms [3] and also because natural images have a granularity greater or equal to 4, and third g=16 (structured random images) to get the optimal behavior of all algorithms.

Global benchmark: Figure 4 provides the execution time in *cpp* (cycles per pixel). Two points can be noticed. First, for $g \in [1, 4]$, the equivalence management algorithm has a major impact. The Suzuki management generates a lost of performance around the percolation threshold – here, for $d \in [40\%, 60\%]$, whereas Union-Find (with/without optimization like DT and PC) does not induce such a dysfunction. Impact is very important for $g = 1$ and still observable for $g = 4$. Second point, LSL algorithms outperform all other algorithms. The fastest version is LSL_{RLE}-Rosenfeld. For g=4, the two curves of LSL-Suzuki are very close (same for LSL-Rosenfeld), whereas, for g=16, the RLE versions become faster than the STD versions: the equivalence management algorithm has a very small impact on performance while RLE compression makes the difference with STD versions and with the other algorithms.

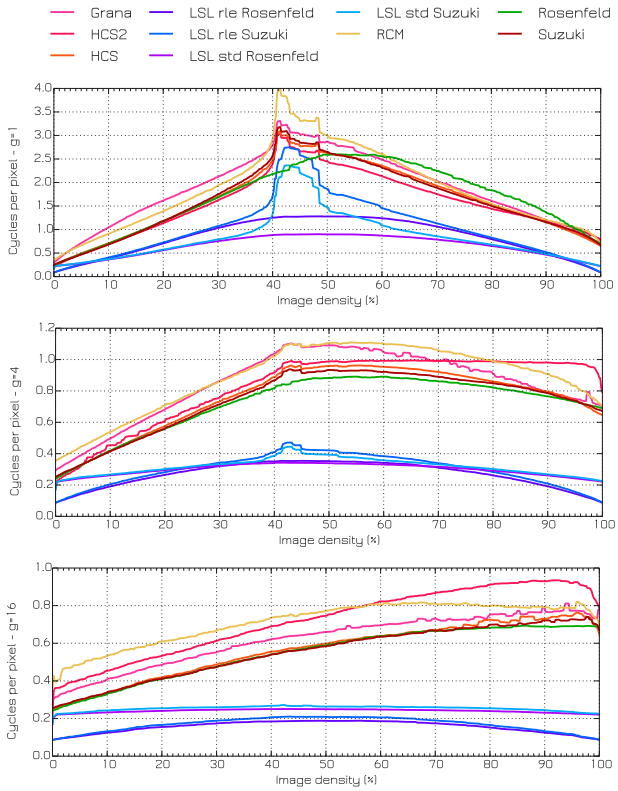

Fig. 4. *cpp* vs density for $g \in \{1,4,16\}$

Speedup: If we look at the speedups between the sequential versions and the parallel ones (tab. 1), we can see than all algorithms have similar and efficient scaling on a 24-core machine. This is remarkable, especially for algorithms who have a non-parallel part that prevent a perfect scaling.

Table 1. speedup according to granularity

	g=1	g=2	g=4	g=8	g=16
LSL_{RLE}-Rosenfeld	22.2	22.7	22.9	22.9	21.6
LSL_{RLE}-Suzuki	19.3	19.0	18.5	18.2	17.6
LSL_{STD}-Rosenfeld	21.5	21.2	21.4	21.4	21.2
LSL_{STD}-Suzuki	18.8	19.3	20.2	20.6	20.6
Suzuki	20.2	19.9	20.3	20.5	20.4
Rosenfeld	19.5	20.6	20.6	20.4	20.5
HCS	21.2	21.3	21.7	21.6	21.1
Grana	19.9	19.6	19.5	19.7	19.8
RCM	19.6	20.6	21.0	20.3	20.2
HCS2	20.1	19.7	20.0	19.0	18.8

Detailed analysis: Figure 5 focuses on the *cpp* of each algorithm step (fist labeling – green, features computation – red and border merging – purple) and highlights three points for $g = 1$. First, the Suzuki equivalence management issue only affects merging step. Second, for Rosenfeld (Union-Find) management, the pyramidal implementation is very efficient for all algorithms and does not represent a significant part of the whole execution time – even with a parallelization on 24 cores. Third, the shape of the curves of the parallel versions is very similar to the sequential ones [3]: the first labeling part takes more time for pixel-based algorithms than for LSL algorithms.

Furthermore, for a given density, there is a better temporal and spatial cache locality for g=4 than for g=1 (and a smaller amount of labels, and shorter decision trees) that leads to a general *cpp*'s decrease. This phenomenon can be observed between figures 5 and 6. We can also observe that while the fraction of FC (related to the total execution time in *cpp*) increases for pixel-based algorithms, it remains approximatively constant for LSL algorithms. Thus, the ratio between them increases with g.

Ratio: If we focus on the ratio between LSL_{RLE}-Rosenfeld and the best competitor (fig. 7), we can see that the ratio increases with g (having in mind that $g = 1$ is the worst configuration for a run-based algorithm like LSL). The average ratio is $\times 1.9$ for $g = 1$, $\times 2.8$ for $g = 4$ and reaches $\times 3.6$ for $g = 16$.

3. CONCLUSION

In this paper, we presented the parallelization of the Light Speed Labeling algorithm for general purpose processor. The results of the benchmarks show that all algorithms have efficient speedup ensuring that implementation and benchmark are fair for all. These results also help the reader who trie to find out the intrinsic differences between the algorithms.

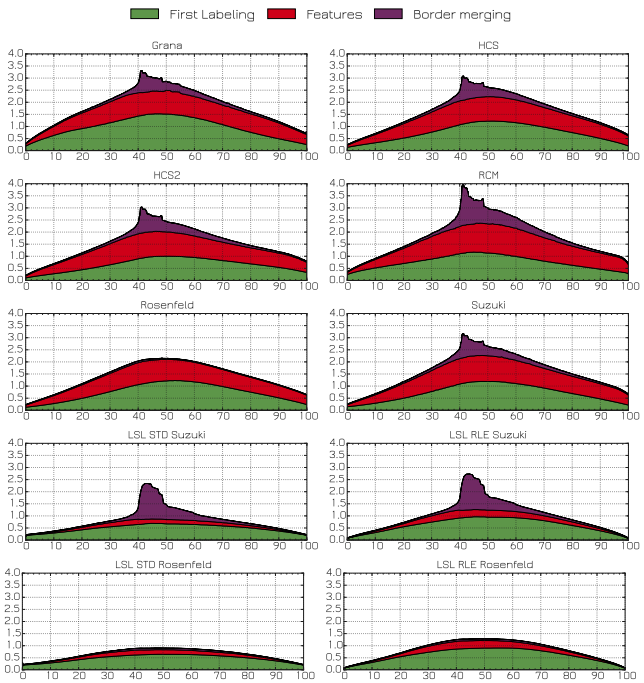

Fig. 5. split *cpp* vs density (%) for $g = 1$

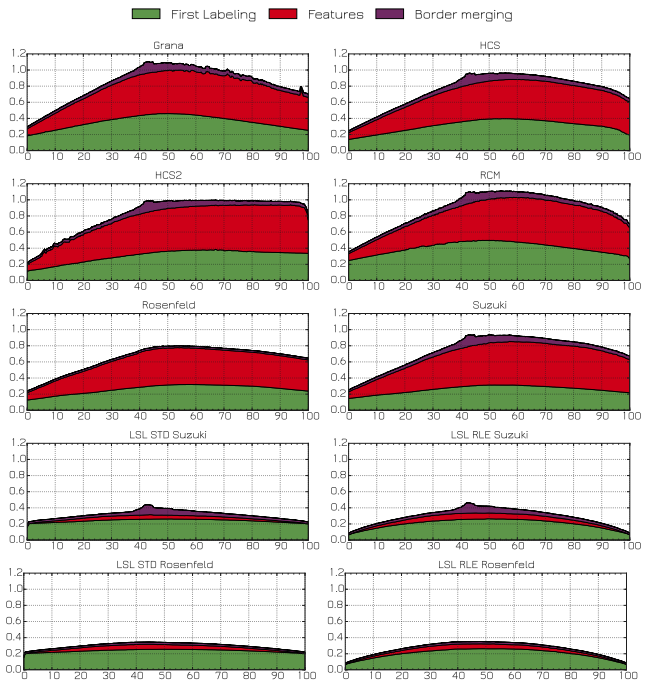

Fig. 6. split *cpp* vs density (%) for $g = 4$

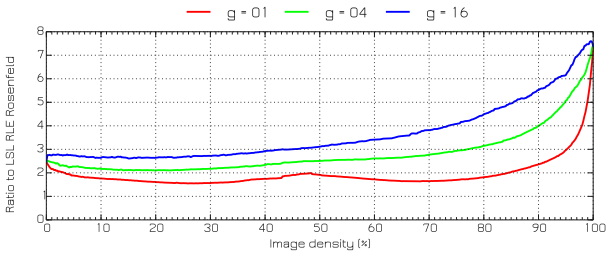

Fig. 7. LSL_{RLE}-Rosenfeld vs best competitor ratio

First, the equivalence management algorithm that does not look significant for sequential benchmark is an issue (for all algorithms) for parallel benchmarks and small granularities ($g \in [1, 4]$): Suzuki's algorithm has a dysfunction around the percolation threshold ($d \in [40\%, 60\%]$). This issue disappears for higher granularities – and by extension – for natural images.

Second, the paper enforces the results of the previous one dealing with sequential comparisons: LSL is *intrinsically* faster than all other pixel-based algorithms as it is fully run-based (then producing less temporary labels than pixel based algorithms) and uses RLE compression to reduce memory accesses. Once the granularity is high enough, the mask of the pixel-based and block-based algorithms has a minor impact on performance, unlike the run-based approach that cluster pixels into segment *before* processing it. With the computation of the bounding box and first statistical moments (results may vary with the selected features), the LSL is $\times 1.9$ faster than its best competitor for random unstructured images. For structured – but still random – images that are close to natural images, the ratio reaches an average factor of $\times 3.6$. As a matter of fact, the parallel LSL_{RLE} execution time, for a 2048×2048 image, is 0.28 ms.

4. REFERENCES

[1] L. Lacassagne and A. B. Zavidovique, "Light speed labeling for RISC architectures," in *IEEE International Conference on Image Analysis and Processing (ICIP)*, 2009.

[2] L. Lacassagne and B. Zavidovique, "Light Speed Labeling: Efficient connected component labeling on RISC architectures," *Journal of Real-Time Image Processing*, vol. 6, no. 2, pp. 117–135, 2011.

[3] L. Cabaret and L. Lacassagne, "What is the world's fastest connected component labeling algorithm ?," in *IEEE International Workshop on Signal Processing Systems (SiPS)*, 2014, pp. 97–102.

[4] C.-W. Chen, Y.-T. Wu, S.-Y. Tseng, and W.-S. Wang, "Parallelization of connected-component labeling on tile64 many-core platform," *Journal of Signal Processing Systems*, vol. 75,2, pp. 169–183, 2013.

[5] L. He, Y. Chao, and K. Suzuki, "A run-based two-scan labeling algorithm," in *ICIAR*. LNCS 4633, 2007, pp. 131–142.

[6] A. Rosenfeld and J.L. Platz, "Sequential operator in digital pictures processing," *Journal of ACM*, vol. 13,4, pp. 471–494, 1966.

[7] U.H. Hernandez-Belmonte, V. Ayala-Ramirez, and R.E. Sanchez-Yanez, "Enhancing ccl algorithms by using a reduced connectivity mask," in *Mexican Conference on Pattern Recognition*, Springer, Ed., 2013, pp. 195–203.

[8] L. He, Y. Chao, and K. Suzuki, "A new two-scan algorithm for labeling connected components in binary images," in *Proceedings of the World Congress on Engineering*, World Congress, Ed., 2012, vol. 2, pp. p1141–1146.

[9] C. Grana, D.Borghesani, and R. Cucchiara, "Fast block based connected components labeling," in *ICIP*. IEEE, 2009, pp. 4061–4064.

[10] L. He, Y. Chao, and K. Suzuki, "An efficient first-scan method for label-equivalence-based labeling algorithms," *Pattern Recognition Letters*, vol. 31, no. 1, pp. 28–35, 2010.